Docket No. SVL920010017US1
Firm No. 0054.0018

METHOD, SYSTEM, AND PROGRAM FOR PERFORMING AN IMPACT ANALYSIS OF PROGRAM STATEMENTS IN AT LEAST ONE SOURCE CODE FILE

RELATED APPLICATION

[0001] This application is related to the copending and commonly assigned patent application entitled "Method, System, And Program For Utilizing Impact Analysis Metadata of Program Statements in a Development Environment", having attorney docket no. SVL920010016US1, filed on the same date herewith, and incorporated herein by reference in its entirety.

BACKGROUND OF THE INVENTION

1. Field of the Invention

[0002] The present invention relates to a method, system, and program for performing an impact analysis of program statements in at least one source code file.

2. Description of the Related Art

[0003] One of the challenges when editing code in a program is to understand the effect of changes to certain lines of code to other parts of the program. In the prior art, software developers use a debugging tool to "step" through the program and determine all the execution paths. Based on the information of the execution characteristics determined by "stepping" through the program, the developer can then analyze the execution path to determine the effect the proposed changes to certain statements may have on the current operation of the program including the program statements to change. This process is manually intensive and is based on the capabilities of the developer to properly observe all the effects the proposed changes will have on the execution characteristics.

[0004] Moreover, a change in one program may affect the operations of another external application by modifying the content of a shared resource, such as a global variable, memory, file, database record, etc. Currently there is no integrated approach for

determining the effects of a proposed change to code on the operation of external applications. In fact, a debugger typically only is aware of the application currently executing, and not the effects on an external application.

[0005] In the current art, the software developer is often unable to ascertain the impact of code modifications to the application including the modified code and to external applications, until errors and bugs are detected. At such point, the developer may then have to spend considerable time tracking down the source of the problem to the code change that was made. This problem is further exasperated if the negative impact of a code change is realized in an external application, where the developer or system manager may be totally unaware of the changes made because they were made in a different application. In such case, time will be spent tracking the source of the error to another application, and then to a particular code change in the external application.

[0006] For these reasons, there is a need in the art for improved software development tools that assist developers in understanding the impact of modifications to a source program.

SUMMARY OF THE PREFERRED EMBODIMENTS

[0007] Provided is a method, system, and program for performing an impact analysis of program statements in a source code file, wherein each program statement has at least one of an input parameter and output parameter. A selection is received of at least one program statement in the source code file. For each selected program statement, a determination is made of program statements in the source code file having as one input parameter one program artifact that is affected by the selected program statement.

[0008] Further provided is a method, system, and program for performing an impact analysis of program statements in a source code file that is one of a plurality of source code files, wherein each program statement has an input parameter and output parameter. Selection is received of at least one program statement in one source code file. For each selected program statement, a determination is made of program statements throughout

the source code files having as one input parameter one program artifact that is affected by the selected program statement.

[0009] In further implementations, the program artifact comprises a variable, Input/Output buffer or file.

[0010] Still further, one program statement has one input parameter that is affected by the selected program statement if the output parameter program artifact of the selected program statement is the input parameter program artifact to the program statement.

[0011] Yet further provided is a method, system, and program for maintaining data on a plurality of source code files. A data store is generated for each source code file and for each program statement in the source code file by generating information on the program statement and generating information on each program artifact referenced as an input parameter in the program statement. Information is further generated on each program artifact referenced as an output parameter in the program statement. The data store is then used to determine program artifacts throughout all of the source code files capable of being affected by any one program statement in any of the source code files.

[0012] The described implementations provide a technique for determining the impact that a change to a selected program statement in one source code file may have on program statements in the source code file including the selected statement and in other source code files. In this way, a software developer may have information on all program statements and program artifacts in one or more source code files that may be affected by changing a selected program statement.

BRIEF DESCRIPTION OF THE DRAWINGS

[0013] Referring now to the drawings in which like reference numbers represent corresponding parts throughout:

FIG. 1 illustrates a computing environment in which aspects of the invention are implemented in accordance with implementations of the invention;

FIG. 2 illustrates data structures maintaining information on program statements and program artifacts throughout source code files in accordance with certain implementations of the invention;

FIG. 3 illustrates logic to process source code files to populate the data structures described with respect to FIG. 2 in accordance with certain implementations of the invention; and

FIGs. 4, 5a, 5b, and 5c illustrate logic to determine the effect a change to selected source code statements may have on statements and program artifacts in source code files in accordance with certain implementations of the invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

[0014] In the following description, reference is made to the accompanying drawings which form a part hereof and which illustrate several embodiments of the present invention. It is understood that other embodiments may be utilized and structural and operational changes may be made without departing from the scope of the present invention.

[0015] FIG. 1 illustrates an enterprise computing system 2 which would be implemented across a network environment including one or more server and client computer systems, comprising any type of computing devices known in the art, e.g., workstations, servers, personal computers, mainframes, hand held computers, laptops, personal information managers (PIMs), telephony devices, etc. A plurality of source code files 4a, 4b...4n are maintained in the enterprise computing system 2 and available to a set of software development tools 6 that are implemented on one or more of the computer systems in the enterprise computing system 2. The source code files 4a, 4b...4n would comprise the source code of application programs used in the enterprise computing system 2 or being developed for deployment in enterprise computing systems.

[0016] The source code files 4a, 4b...4n may also include a job control language (JCL) program that includes program statements that call and execute application programs in the system and associates logical and physical entities. For instance, within an

application, a logical data set name may be used to reference or call a physical data set. The JCL program provides the association of the logical data set to the physical data set. In fact, the same logical name may be used to reference different physical data sets. The JCL association of physical and logical entities is examined when performing the impact analysis to determine how code modifications can affect a physical data set and, in turn, the statements that reference that physical data set. A set of JCL statements may specify the input data sets (files) to access, output data sets to create or update, resources to allocate for the job, and the programs that are to run using the defined input and output data sets.

[0017] The source code files 4a, 4b...4n may be stored in specific file directories or maintained and managed by a source control management system known in the art, such as the International Business Machine Corporation's (IBM) Software Configuration and Library Manager (SCLM), Microsoft Visual SourceSafe, etc.** The source code files 4a, 4b...4n may comprise component files of one or more application programs, where each application program is comprised of one or more source code files. A source control management system provides a library management system of source code and controls access and provides locking to prevent developer conflicts. An example of a source control management system that may be used to manage developer access to the source code files 4a, 4b...4n is described in the IBM publication "Using ISPF/SCLM for Automated and Controlled Software Development", IBM publication no. SG24-4843-00 (IBM Copyright, October 1996), which publication is incorporated herein by reference in its entirety.

[0018] The enterprise computing system 2 further includes a set of software development tools 6 deployed on one or more systems to assist software developers in modifying and updating the source programs 4a, 4b...4n. The software development tools 6 include an inventory collection program 8 that scans and analyzes all the source programs 4a, 4b...4n and generates an artifact database 12 providing metadata on all the program statements and program artifacts referenced in the source programs 4a, 4b...4n. A program artifact may comprise a program statement, program variable, Input/Output

buffers, files, data sets, or any other data structure known in the art manipulated by a computer program. Once the inventory collection program 8 generates the artifact database 12, then an impact analysis tool 10 may be invoked to analyze any block of statements in any of the source programs 4a, 4b...4n. This analysis of the block of statements would generate a report on all the program statements and program artifacts across all of the source programs 4a, 4b...4n that could possibly be affected by any change to the selected block of statements.

[0019] FIG. 2 illustrates a relational database implementation of the artifact database 12 maintaining the artifact elements gathered from the source code files 4a, 4b...4n. The artifact database 12 includes a program statement table 20, a variable table 40, an I/O buffer table 50, and a file table 60. The program statement table 20 includes one record 22 for each program statement in each source code file 4a, 4b...4n. Each program statement record 22 includes a unique statement key 24, a program identifier (ID) 26 uniquely identifying the source code file 4a, 4b...4n including the statement, the line number 28 in the source code file 4a, 4b...4n including the statement, and the statement command name or verb 30. If a line of code includes multiple statements, i.e., verb and input and output parameters, then multiple statement records 22 would be generated for that line of code. The program ID 26 may comprise the file name of the source code file 4a, 4b...4n or, if the program is maintained in a source control management system, then the program ID 26 may comprise the identifier of source code file 4a, b...n in the source control management system library.

[0020] The variable table 40 includes one variable record 42 for each instance a variable is referenced in a program statement in any of the source code files 4a, 4b...4n. Each variable record 42 includes a variable name 44, the unique statement key 46 of the program statement in which the variable was referenced, and an input/output flag 48 indicating whether the variable was input to the program statement or the output/target of the program statement and possibly modified. The I/O buffer table 50 includes an I/O buffer record 52 for each instance of a read or write operation to an I/O buffer in a program statement in any of the source code files 4a, 4b...4n. Each I/O buffer record 52

includes the identifier of the allocated I/O buffer 54, the unique statement key 56 of the program statement operating on the I/O buffer, and a read/write flag 58 indicating whether the program statement performed a read or write with respect to the I/O buffer. The file table 60 includes a file record 62 for each instance of a read or write operation to a file in a program statement in any of the source code files 4a, 4b...4n. Each file record 62 includes the identity of the effected file 64, the unique statement key 66 of the statement operating on the file, an Input/Output flag 67 indicating whether the statement performed an input or output with respect to the source code file 4a, 4b...4n, and a read/write flag 68 indicating whether the statement performed a read or write with respect to the file.

[0021] FIG. 3 illustrates logic implemented in the inventory collection program 8 to populate the artifact database 12 with program artifact metadata. The logic of FIG. 3, as well as the logic of FIGs. 4, 5a, 5b, and 5c would be executed by one or more computer systems on which the software development tools are installed. Control begins at block 100 by accessing all the source code files 4a, 4b...4n subject to the analysis, either through the file directory or by accessing the source code files 4a, 4b...4n through a control management system. A loop is then performed at block 102 through block 126 for each source code file *i* to consider. At blocks 104 through 124, the inventory collection program 8 performs a loop for every line *j* in the source code file *i*, from the top to bottom of the code. The inventory collection program 8 scans (at block 106) line *j* for a program statement. If there are multiple program statements in the line, then the steps 108 through 122 will be performed for each statement in the line, where a statement comprises a command verb and an input and output program artifact, e.g., variable, I/O buffer, file, etc., acted upon by the program statement. A program statement record 22 is generated including fields having: the unique statement key 24, which would be generated with the record; the program identifier 26; line number *j* 28; and the command verb 30, which may comprise a command specifying a particular operation to be performed on the input and output parameters or a call to a subroutine within the source code file *i* or to a routine in an external source code file 4a, 4b...4n. Block 110 then

inserts the generated program statement record 22 is then inserted into the program statement table 20 in the artifact database 12.

[0022] For each variable referenced as an input or output parameter in the located program statement, the inventory collection program 8 generates (at block 112) a variable record 42 including: the variable name 44, the unique statement key 46 identifying the program statement referencing the variable as an input or output parameter, and sets the Input/Output flag 48 to indicate whether the variable is an input parameter or output parameter to the statement. The generated variable record 42 is inserted (at block 114) into the variable table 40. For each I/O buffer referenced in the program statement, an I/O buffer record 52 is generated (at block 116) including an identifier of the I/O buffer 54, the statement key 56 of the program statement operating on the I/O buffer as a parameter; and a read/write flag 58 indicating whether the program statement performs a read or write operation with respect to the I/O buffer. The generated I/O buffer record 52 is then inserted (at block 118) into the I/O buffer table 50. For each file referenced in the program statement, a file record 62 is generated (at block 120) including an identifier of the file 64 (such as the physical location of the file in the file system), the statement key 66 of the program statement operating on the file, an Input/Output flag 67, and a read/write flag 68 indicating whether the statement performs a read or write operation with respect to the file. The generated file record 62 is then inserted into the file table 60.

[0023] At block 124, control proceeds back to block 104 to process the next line of the source code file *i*. After processing all the lines in the source code file *i*, control proceeds (at block 126) back to block 102 to process the next source code file 4a, 4b...4n being considered. The result of the logic of FIG. 3 is an artifact database 12 populated with metadata about all statements in the source code files 4a,b ...n and all program artifacts that are affected by the statements. The inventory collection tool 8 may update the artifact database 12 if one source code file 4a, 4b...4n is changed. This update may be performed by removing all records that reference the updated source code file 4, b...n and then performing the logic of FIG. 3 to add the metadata on all the program artifacts back to the artifact database 12.

[0024] FIGs.4, 5a, 5b, and 5c illustrate logic implemented in the impact analysis tool 10 to query the artifact database 12 to determine all program artifacts across all the source code files 4a, 4b...4n that may be affected by a change to a selected one or more program statements in one source code file 4a, 4b...4n. As discussed, the user may select one or more lines of code in one source code file 4a, 4b...4n for the impact analysis tool 10 to consider. Control begins at block 150 in FIG. 4 upon receiving a call to the impact analysis tool 10 and selection of one or more program statements to subject to impact analysis. For each of the program statements subject to analysis, a loop is performed at blocks 152 through 156. At block 154, an impact analysis routine is called with statement *i* as an input parameter to determine all program artifacts directly and indirectly affected by statement *i* FIGs. 5a, 5b, and 5c illustrate the logic of the impact analysis routine.

[0025] At block 200 in FIG. 5a. a call is made to the impact analysis routine to analyze an input program statement. If (at block 202) the input program statement does not call another routine within the same or external source code file 4a, 4b...4n and if (at block 204) the input program statement affects a program artifact, then a determination is made at block 206 of whether the input statement modifies a variable. An input program statement would not affect a program artifact if it does not modify the program artifact, such as the case if the input program statement specifies that data be sent to a printer or other output device that does not modify any other variable, file or I/O buffer. If the input program statement modifies an input variable (from the yes branch of block 206), then the impact analysis routine would query (at block 208) the variable table 40 to determine all statements that reference the modified variable, i.e., have the same variable name 44, as an input parameter, i.e., the Input/Output flag 48 is set to indicate input. An output data listing of all statements that reference the modified variable as an input parameter is produced (at block 210). At blocks 212 through 216, for each statement *j* referencing a modified variable as an input variable, the impact analysis routine makes a recursive call (at block 214) to the impact analysis routine, which begins at block 200, with statement *j* as an input variable to determine all program artifacts that statement *j*,

which has as an input parameter a variable affected directly by the program statement *i* or indirectly by another program statement affected by the program statement *i*. After control is returned from the nested call to the impact analysis routine, control proceeds to block 218 to return control to the point in the program where the call to the impact analysis routine was made, which may be at block 154 in FIG. 4 or within a nested call within FIGs. 5a, 5b, 5c from where the impact routine was called. Control is returned with information output indicating all the directly and indirectly affected program statements.

[0026] If (at block 206) the input program statement does not modify a variable and if (at block 250) the input program statement does write to a target I/O buffer, then the impact analysis routine queries (at block 252) the I/O buffer table 50 to determine all I/O buffer records 52 identifying the target I/O buffer and that indicate in the read/write flag 58 that the statement reads from the I/O buffer. From the determined I/O buffer records 52, the impact analysis routine determines and generates output data (at block 254) of all statements read from the target I/O buffer as indicated from the statement key 56 value in the determined I/O buffer records 52. The routine generates output data indicating all program statements that read from the affected I/O buffer. A loop is then performed at blocks 256 through 260 for each program statement reading from an I/O buffer affected by the input program statement. At block 258, the impact analysis routine is called (at block 200 in FIG. 5a) with one determined program statement that reads from the affected I/O buffer as an input parameter to determine all further program artifacts affected by the program statement reading from the affected I/O buffer.

[0027] If (at block 250) the input program statement does not write to an I/O buffer, then the impact analysis routine assumes that the input statement writes to a file (at block 280), which is the last program artifact to consider in the logic of FIGs. 5a, 5b, and 5c. Those skilled in the art will appreciate that the logic of FIGs. 5a, 5b, and 6c can be extended to consider the effect on program artifacts other than variables, I/O buffers, and files. At block 282, the impact analysis routine determines the physical location of the file in the file system if the statement operates on a logical file name. Such physical

location may be provided by a declaration in the source code file *i* or an external job control language program defining a logical name associated with a physical file. At block 284, the impact analysis routine queries the file table 60 to determine all file records 62 identifying the target file modified by the statement and that indicate in the read/write flag 68 that the statement reads from the file. From the determined file records 62, the impact analysis routine determines statements that read from the target file from the statement key 66 value in the determined file records 62. The routine generates output data indicating all statements that read from the affected I/O buffer. A loop is then performed at blocks 286 through 290 for each program statement reading from an I/O buffer affected by the input statement. At block 288, the impact analysis routine is called (at block 200 in FIG. 5a) with one determined program statement that reads from the affected file as an input parameter to determine all further program artifacts affected by the program statement reading from the affected file. From blocks 260 or 292, control proceeds to block 218 to return to the point in the program execution where the call to the impact analysis routine was made.

[0028] If (at block 204) the input statement does not affect a program artifact, such as the case if the statement writes data to a printer, display monitor, etc., then control proceeds to block 218 to return control to the point in the program execution where the call to the impact analysis routine was made. If (at block 202) the input statement calls a routine within the same or another source code file 4a, 4b...4n, then control proceeds to block 300 in FIG. 5c to determine the impact from modifying a program statement including a call to another program routine comprised of one or more program statements..

[0029] With respect to FIG. 5c, at block 300, the impact analysis routine determines the source code file including the called routine. The program statement table 20 is then queried (at block 302) to determine all program statement records 22 that reference the input parameter of the called routine as input. This operation is performed to determine all statements in the called routine that may use the routine input parameter to affect other program artifacts. This determination of statements in the called routine that use

the input parameter as input can be determined by a query of the variable table 40, I/O buffer table 50, and file table 50 of all table records 42, 52, and 62 having a statement key 46, 56, or 66 identifying a statement in the called source code file 4a, 4b...4n (previously determined by a query of the program statement table 20 for all program statement records 22 including the called source code file as the program ID 26 and whose input/output flag 48, 58, or 68 indicates that the input parameter is input to the program statement. Output is then generated (at block 304) of all statements that use the called input parameter as input. For each determined statement *j* referencing the input parameter as input, a loop is performed at blocks 306 through 310 where the impact analysis routine is called (at block 308) to determine all program artifacts affected by statement *j*.

[0030] The impact analysis program than determines at blocks 312 through 320 the impact on all program statements that reference the output parameter of the called routine as input, i.e., that may use the parameter affected by the called routine. At block 312, the impact analysis routine queries the tables 20, 40, 50, 60 to determine all program statement records 22 that reference the output parameter of the called routine as input. This can be determined by a query of the variable table 40, I/O buffer table 50, and file table 50 of all table records 42, 52, and 62 having a statement key 46, 56, or 66 identifying a statement in the called source code file 4a, 4b...4n (previously determined by a query of the program statement table 20 for all program statement records 22 including the called source code file as the program ID 26) and whose input/output flag 48, 58, 68 indicates that the output parameter to the routine is input to the statement. Output is then generated (at block 314) of all routine statements that use the called output parameter as input. For each determined statement *j* referencing the routine output parameter as input, a loop is performed at blocks 316 through 320 where the impact analysis routine is called (at block 318) to determine all program artifacts affected by statement *j*.

[0031] The resulting output of a call to the impact analysis tool 10 to analyze one or more statements in a source code file 4a, b..n is a list of all program statements that

reference an input parameter that may be affected by a modification to the analyzed statement. This output lists statements referencing as input a program artifact that is directly affected by the analyzed statement as well as program artifacts indirectly affected, i.e., statements that reference an input program artifact that was affected by a statement referencing an input parameter affected by the analyzed statement, etc. From block 320, control proceeds to block 218 in FIG. 5a to return control to the point in the program from where the call to the impact analysis routine was generated.

[0032] In further implementations, the impact analysis tool 10 may also generate extended information explaining the source of the affect on a listed program statement. As discussed, at block 210, 254, 288, and 314 in FIGs. 5a, 5b, and 5c, the impact analysis tool 10 generates output data listing all program statements that reference a modified program artifact as an input statement. Thus, when generating information at this point, the impact analysis tool 10 may further generate extended information for an affected program statement identifying the previous program statement that modifies the program artifact used as the input statement by the affected program statement and also provide information on the exact transformation performed by the previous program statement that modifies the program artifact used as the input statement.

[0033] After the impact analysis tool 10 generates the output of affected source code files and program statements therein, the software developer may review the output listing of affected statements across all files 4a, 4b...4n to determine what impact the proposed changes will have on all the different application programs comprised of one or more of the source code files 4a, 4b...4n that include the affected statements.

[0034] The resulting output of a call to the impact analysis tool 10 to analyze one or more statements in a source code file 4a, b..n is a list of all program statements that reference an input parameter that may be affected by a modification to the analyzed statement. This output lists statements referencing as input a program artifact that is directly affected by the analyzed statement as well as program artifacts indirectly affected, i.e., statements that reference an input program artifact that was affected by a statement referencing an input parameter affected by the analyzed statement, etc. The

software developer may review the output listing of affected statements across all files 4a, 4b...4n, which are part of one or more application programs, to determine what impact the proposed changes will have on all the different programs, including the application program source code file having the affected statement. The related application entitled "Method, System, and Program for Utilizing Impact Analysis Metadata of Program Statements in a Development Environment", having docket no. SVL920010016US1, incorporated by reference above, provides further details on how the impact analysis tool 10 may be invoked and use as part of an integrated development environment.

Additional Implementation Details

[0035] The preferred embodiments may be implemented as a method, apparatus or article of manufacture using standard programming and/or engineering techniques to produce software or code. The term "article of manufacture" as used herein refers to code or logic implemented in a computer readable medium (e.g., magnetic storage medium (e.g., hard disk drives, floppy disks, tape, etc.), optical storage (CD-ROMs, optical disks, etc.), volatile and non-volatile memory devices (e.g., EEPROMs, ROMs, PROMs, RAMs, DRAMs, SRAMs, firmware, programmable logic, etc.). Code in the computer readable medium is accessed and executed by a processor. The code in which preferred embodiments are implemented may further be accessible through a transmission media or from a file server over a network. In such cases, the article of manufacture in which the code is implemented may comprise a transmission media, such as a network transmission line, wireless transmission media, signals propagating through space, radio waves, infrared signals, etc. Of course, those skilled in the art will recognize that many modifications may be made to this configuration without departing from the scope of the present invention, and that the article of manufacture may comprise any information bearing medium known in the art.

[0036] In the described implementations of FIG. 2, the database comprised a relational database having a separate program statement table and a separate table for each

considered program artifact. Those skilled in the art will appreciate that the statement and program artifact data in the tables of FIG. 2 may be implemented in an any type of data store known in the art, such as a relational database having a different table and record structure, a database system other than a relational database as described herein (e.g., an object oriented database), a flat file or one or more files or data structures.

[0037] In further implementations, the records in the artifact database 12 may include additional fields concerning more detail on the format and structure of the program artifacts and their relationship to one another. Additionally, the tables and fields in the tables described with respect to FIG. 2 may have additional fields or have the data implemented in tables in a different manner.

[0038] The described implementations discussed an impact analysis performed with respect to program artifacts comprising an affected I/O buffer, file and variable program. In further implementations, additional program artifacts or data structures known in the art may be subject to the impact analysis.

[0039] The described implementations discussed an impact analysis of a selected statement on program statements in the source code file including the selected statement and in other source code files. Additionally, the impact analysis of the present invention may be used to determine the impact of a selected program statements on program statements within the source code file including the selected program statements, and not in any other source code files.

[0040] The described implementations may be utilized to maintain metadata on the program artifacts on any number of designated programs. Further, the source code files 4a, 4b...4n and software development tools described herein do not have to be implemented in an enterprise computing system. Additionally, the source code files 4a, 4b...4n and software development tools 6 may be maintained at a single computer workstation, where the tools 6 are deployed when modifying any of the source code files 4a, 4b...4n

[0041] The described logic of FIGs. 3, 4, 5a, 5b, and 5c describes specific operations occurring in a particular order. In alternative implementations, certain of the logic

operations may be performed in a different order, modified or removed. Morever, steps may be added to the above described logic and still conform to the described implementations. Further, operations described herein may occur sequentially or certain operations may be processed in parallel.

[0042] The foregoing description of the preferred embodiments of the invention has been presented for the purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise form disclosed. Many modifications and variations are possible in light of the above teaching. It is intended that the scope of the invention be limited not by this detailed description, but rather by the claims appended hereto. The above specification, examples and data provide a complete description of the manufacture and use of the composition of the invention. Since many embodiments of the invention can be made without departing from the spirit and scope of the invention, the invention resides in the claims hereinafter appended.

**Microsoft and Visual SourceSafe are trademarks of Microsoft Corporation in the United States, other countries, or both; IBM is a registered trademark of International Business Machines Corporation in the United States, other countries, or both.